Exploitation d'une Base de Données
SAE 2.04

Sommaire

LATIF Marya | CABO India | BAOUCHI Aboubakar

Modélisation de Données

Cahier des charges

Contexte

Lors des années scolaires, de la primaire aux études supérieures, des contrôles sont réalisés par les élèves/étudiants. Une note est attitrée à chacun des contrôles. Plusieurs notes ensemble forment une moyenne qui est assignée à un module. De plus, la moyenne de différentes moyennes forme la moyenne d'une compétence. Cette dernière est très importante car elle permet de savoir si l'étudiant possédant ladite moyenne valide ou non son année. Tout cela montre l'importance portée sur le stockage et la gestion des notes d'un étudiant. C'est sur quoi le projet porte : la création et gestion de la base de données.

Objectif

- ➢ Étudier un modèle de données pour mettre en place une base de données de gestion des notes des étudiants en BUT
- ➢ Étudier et mettre en œuvre la gestion des données dérivées : relevé de notes, bilans, etc.
- ➢ Étudier et mettre en œuvre des restrictions d'accès à ces données : étudiant, enseignant, responsable de matière, etc

Périmètre

- ➢ Dans la base de donnée
    - ○ Étudiants
    - ○ Enseignants
    - ○ Responsables des modules
- ➢ Qui peut voir la base de donnée
    - ○ Étudiants
    - ○ Enseignants
    - ○ Responsables des modules
    - ○ Directeurs
    - ○ Personnel supérieur

Besoin

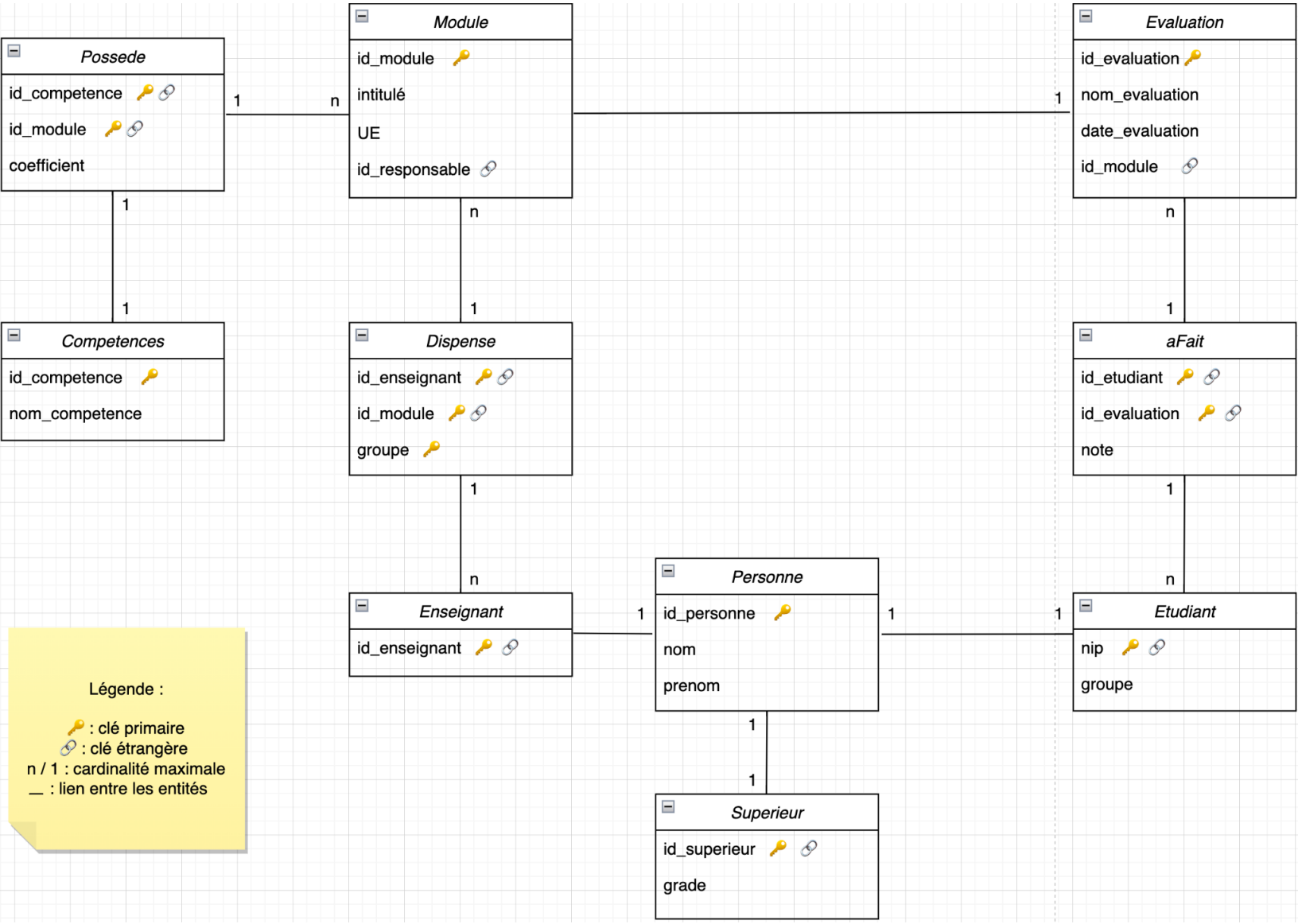
- ➢ Il faut tout d'abord modéliser notre base de donnée à l'aide d'un logiciel pour avoir une idée des tables à créer
- ➢ Réaliser le script de la création de la base de données à partir du modèle réalisé en utilisant un des langages étudiés (PostgreSql)

- Définir des contraintes de restrictions sur les données d'un utilisateur

Contraintes

- **Délais :** 23 Mai 2023
- **Technique**
  - Logiciel
    - Draw.io
    - pgAdmin 4
  - Base de Donnée
    - PostgreSql
  - Matériel
    - Ordinateur portable

Modèle de la Base de Données



Définir les règles de gestion de ces données et leurs mises en œuvre par des procédures stockées.

Le responsable du module a le droit de :

- Ajouter/Modifier/Supprimer des notes : Procédure Trigger
  - trigger_note() avec modif_note() sur la table aFait
- Ajouter/Modifier/Supprimer des évaluations : Procédure Trigger
  - modif_eval() avec modif_eval() sur la table evaluation
- Consulter des notes : Procédure
  - notes_de(nip int)
  - bulletin_de(_nip int)
- Consulter le classement des élèves : Procédure
  - classement()
- Consulter les moyennes : Procédure
  - moyenne_de(nip int)

L'étudiant a le droit de :

- Consulter ses notes : Procédure
  - mes_notes()
  - mon_bulletin()
- Consulter sa moyenne : Procédure
  - mes_moyennes()
- Consulter son classement : Procédure
  - mon_classement()

Script de création de la Base de Données

```
CREATE TABLE Personne(

   id_personne INTEGER PRIMARY KEY,

   nom_personne VARCHAR,

   prenom_personne VARCHAR

);

CREATE TABLE Enseignant(

   id_enseignant INTEGER PRIMARY KEY REFERENCES Personne(id_personne)
```

```
);

CREATE TABLE Superieur(

   id_superieur INTEGER PRIMARY KEY REFERENCES Personne(id_personne),

   grade VARCHAR

);

CREATE TABLE Etudiant(

   NIP INTEGER PRIMARY KEY REFERENCES Personne(id_personne),

   groupe VARCHAR

);

CREATE TABLE Module(

   id_module INTEGER PRIMARY KEY,

   intitule VARCHAR,

   UE VARCHAR,

   id_enseignant INTEGER REFERENCES Enseignant(id_enseignant)

);

CREATE TABLE Competences(

   id_competence INTEGER PRIMARY KEY,

   nom_competence VARCHAR

);

CREATE TABLE Dispense(

   id_enseignant INTEGER REFERENCES Enseignant(id_enseignant),

   id_module INTEGER REFERENCES Module(id_module),

   groupe VARCHAR,

   PRIMARY KEY(id_enseignant, id_module, groupe)
```

```
);

CREATE TABLE Possede(

   id_competence INTEGER REFERENCES Competences(id_competence),

   id_module INTEGER REFERENCES Module(id_module),

   coef DECIMAL(4,2),

   PRIMARY KEY(id_competence, id_module)

);

CREATE TABLE Evaluation(

   id_eval SERIAL PRIMARY KEY,

   nom_eval VARCHAR,

   coef_eval DECIMAL(4,2),

   date_eval DATE,

   id_module INTEGER REFERENCES MODULE(id_module)

);

CREATE TABLE aFait(

   id_eval INTEGER REFERENCES Evaluation(id_eval),

   id_etudiant INTEGER REFERENCES Etudiant(NIP),

   PRIMARY KEY(id_eval, id_etudiant),

   note DECIMAL(4,2)

);
```

Visualisation de Données

Ensemble de données dérivées à visualiser

Moyenne module étudiant : PROCÉDURE #1

- Relevé de note : PROCÉDURE #2
- Bulletin étudiant en fonction du groupe : PROCÉDURE #3
- Contrôles : VIEW
- Compétences : VIEW
- Moyenne dans chaque groupe : VIEW
- Classement : PROCÉDURE #4

Procédures et Vues pour accéder à ces données

Pour faciliter et alléger l'implémentation des fonctions qui nécessitent de calculer des moyennes nous avons décidé de créer des fonctions intermédiaires tel que moyenne, moyenne_max, moyenne_min, moyenne_generale et moyenne_competence.

```
-- Moyenne de l'étudiant sur le module  #1

CREATE OR REPLACE FUNCTION moyenne(nip_ int, module_ int) RETURNS decimal AS

$$

DECLARE

   curseur_moy CURSOR(nip int, id_mod int) FOR

       SELECT coef_eval, note FROM aFait

       JOIN Evaluation USING(id_eval)

       JOIN Module USING(id_module)

           WHERE id_etudiant = nip AND id_module = id_mod;
```

```
   coef decimal;
   note decimal;
   sum_note decimal;
   sum_coef decimal;
BEGIN
   OPEN curseur_moy(nip_, module_);
   sum_note:=0;
   sum_coef:=0;
   LOOP
       FETCH curseur_moy INTO coef, note;
       EXIT WHEN NOT FOUND;
       sum_note:= sum_note + (coef*note);
       sum_coef:= sum_coef + coef;
   END LOOP;
   CLOSE curseur_moy;
   IF sum_coef!=0 THEN
       RETURN ROUND(sum_note/sum_coef,2);
   END IF;
   RETURN ROUND(0,2); -- Si l'élève n'a pas de note il a 0
END
$$ LANGUAGE plpgsql;
-- Moyenne la plus haute sur le module
CREATE OR REPLACE FUNCTION moyenne_max (module_ integer) RETURNS decimal AS
$$
```

```
DECLARE
   curseur CURSOR FOR SELECT nip FROM Etudiant;
   nip_ int;
   max decimal:=0;
BEGIN
   OPEN curseur;
   LOOP
       FETCH curseur INTO nip_;
       EXIT WHEN NOT FOUND;
       IF max < moyenne(nip_, module_) THEN
           max:= moyenne(nip_, module_);
       END IF;
   END LOOP;
   CLOSE curseur;
   RETURN max;
END
$$ LANGUAGE plpgsql;
-- Moyenne la plus basse sur le module
CREATE OR REPLACE FUNCTION moyenne_min (module_ integer) RETURNS decimal AS
$$
DECLARE
   curseur CURSOR FOR SELECT nip FROM Etudiant;
   nip_ int;
   min decimal;
```

```
BEGIN

   OPEN curseur;

   min:= moyenne(12200955, module_);

   LOOP

       FETCH curseur INTO nip_;

       EXIT WHEN NOT FOUND;

       IF min > moyenne(nip_, module_) THEN

           min:= moyenne(nip_, module_);

       END IF;

   END LOOP;

   CLOSE curseur;

   RETURN min;

END

$$ LANGUAGE plpgsql;

-- Moyenne sur la competence

CREATE OR REPLACE FUNCTION moyenne_competence(nip_ int, comp int) RETURNS DECIMAL AS

$$

DECLARE

   curseur_comp CURSOR(compe int) FOR SELECT * FROM Possede

       WHERE id_competence = compe;

   competence int;

   module int;

   coef decimal;

   sum_coef decimal:=0;
```

```
   sum_moy decimal:=0;
BEGIN
   OPEN curseur_comp(comp);
   LOOP
       FETCH curseur_comp INTO competence, module, coef;
       EXIT WHEN NOT FOUND;
       sum_moy:=sum_moy+moyenne(nip_, module)*coef;
       sum_coef:=sum_coef+coef;
   END LOOP;
   CLOSE curseur_comp;

   IF sum_coef!=0 THEN
       RETURN ROUND(sum_moy/sum_coef,2);
   END IF;
   RETURN 0;
END
$$ LANGUAGE plpgsql;
-- Moyenne générale
CREATE OR REPLACE FUNCTION moyenne_general(nip int) RETURNS DECIMAL AS
$$
DECLARE
   curseur_moyG CURSOR FOR SELECT id_competence FROM Competences;
   comp int;
   sum_comp decimal:=0;
```

```
   nb int;
BEGIN
   SELECT COUNT(id_competence) INTO nb FROM Competences;
   OPEN curseur_moyG;
   LOOP
       FETCH curseur_moyG INTO comp;
       EXIT WHEN NOT FOUND;
       sum_comp:= sum_comp+moyenne_competence(nip, comp);
   END LOOP;
   CLOSE curseur_moyG;
   RETURN ROUND(sum_comp/nb,2);
END
$$ LANGUAGE plpgsql;
```

Les fonctions qui suivent sont celles qui affichent les données

```
-- Visualisation de toutes les competences
CREATE OR REPLACE VIEW competence AS
   SELECT competence, nom_personne, prenom_personne, intitule, UE
   FROM Module, Personne, Enseignant
       WHERE id_enseignant = id_personne;
-- Visualisation de tous les controles
CREATE OR REPLACE VIEW controle AS
   SELECT intitule, coef_eval, aFait, MAX(note), AVG(note), MIN(note)
   FROM Module JOIN Evaluation USING(id_module), aFait;
--Visualisation de toutes les moyennes dans chaque competence en fonction des groupes
```

```
CREATE VIEW moyenne_groupe_competence

AS

   SELECT comp, groupe, ROUND(AVG(moy),2)

   FROM Etudiant JOIN bulletin(groupe) ON nip = nip_

       GROUP BY comp, groupe

       ORDER BY groupe;

-- Visualisation du bulletin de chaque élève d'un groupe #3

CREATE OR REPLACE FUNCTION bulletin (grp varchar, out nip_ integer, out nom varchar,
out prenom varchar, out groupe_ varchar, out comp varchar, out moy decimal) RETURNS
SETOF record AS

$$

DECLARE

   curseur CURSOR FOR SELECT DISTINCT nip, nom_personne, prenom_personne, groupe,
nom_competence, id_competence

   FROM Etudiant JOIN Personne ON nip = id_personne

   JOIN aFait ON id_etudiant = nip

   JOIN Evaluation USING(id_eval)

   JOIN Possede USING(id_module)

   JOIN Competences USING(id_competence)

   ORDER BY nom_personne;

   id_comp int;

BEGIN

   OPEN curseur;

   LOOP

       FETCH curseur into nip_, nom, prenom, groupe_, comp, id_comp;
```

```
        EXIT WHEN NOT FOUND;

        IF groupe_ = grp THEN

            moy:=moyenne_competence(nip_, id_comp);

            RETURN NEXT;

        END IF;

    END LOOP;

    CLOSE curseur;

END

$$ LANGUAGE plpgsql;

-- Visualisation des moyennes de chaques élèves d'un groupe dans chaque module

CREATE OR REPLACE FUNCTION moyenne_etudiant(

    grp varchar,

    out nip_ int,

    out nom varchar,

    out prenom varchar,

    out groupe_ varchar,

    out module varchar,

    out min_moyenne decimal,

    out moyenne_etudiant decimal,

    out max_moyenne decimal

    ) RETURNS setof record AS

$$

DECLARE

    curseur_etud CURSOR FOR
```

```
      SELECT DISTINCT id_etudiant, nom_personne, prenom_personne, groupe, intitule,
id_module

      FROM Personne

      JOIN Etudiant ON id_personne = NIP

      JOIN aFait ON NIP = id_etudiant

      JOIN Evaluation USING(id_eval)

      JOIN Module USING(id_module)

      ORDER BY nom_personne;



   idModule int;

BEGIN

   OPEN curseur_etud;

   LOOP

      FETCH curseur_etud INTO nip_, nom, prenom, groupe_, module, idModule;

      EXIT WHEN NOT FOUND;

      IF groupe_ = grp THEN

          moyenne_etudiant:=moyenne(nip_, idModule);

          min_moyenne:=moyenne_min(idModule);

          max_moyenne:=moyenne_max(idModule);

          RETURN NEXT;

      END IF;

   END LOOP;

   CLOSE curseur_etud;

END

$$ LANGUAGE plpgsql;
```

```
-- Visualisation du relevé de note de l'élève entré en paramètre  #2

CREATE or replace FUNCTION releve_de_note(inout nip_ integer, out nom varchar, out
prenom varchar,out module varchar,out nom_controle varchar,out meilleure_note decimal,
out note_ decimal, out pire_note decimal)

RETURNS SETOF RECORD AS

$$

   DECLARE

       etud_note CURSOR(nip_ int) FOR SELECT id_etudiant, nom_personne,
prenom_personne, intitule, nom_eval, note, aFait.id_eval

       FROM Personne JOIN aFait ON id_personne = id_etudiant JOIN Evaluation
USING(id_eval) JOIN Module USING(id_module)

           WHERE id_etudiant = nip_;

       idEval int;

   BEGIN

       open etud_note(nip_);

       LOOP

           FETCH etud_note INTO nip_, nom, prenom, module, nom_controle, note_,
idEval;

           EXIT WHEN NOT FOUND;

           SELECT MIN(note) INTO pire_note FROM aFait WHERE id_eval = idEval;

           SELECT MAX(note) INTO meilleure_note FROM aFait WHERE id_eval = idEval;

           RETURN NEXT;

       END LOOP;

       close etud_note;

   END

$$ LANGUAGE plpgsql;
```

```
-- Visualisation du classement général #4

CREATE OR REPLACE FUNCTION classement (out nip_ int, out prenom varchar, out nom
varchar, out moyenne decimal, out classement int)

RETURNS SETOF RECORD AS

$$

   DECLARE

       curs_clsmt CURSOR FOR

       SELECT DISTINCT id_personne, prenom_personne, nom_personne,
moyenne_general(id_personne) as moy

           FROM Personne

               JOIN Etudiant ON nip = id_personne

               ORDER BY moy desc;

   BEGIN

       classement :=1;

       OPEN curs_clsmt;

       LOOP

           FETCH curs_clsmt INTO nip_, prenom, nom, moyenne;

           EXIT WHEN NOT FOUND;

           RETURN NEXT;

           classement:=classement+1;

       END LOOP;

   END

$$ LANGUAGE plpgsql;
```

Restrictions d'accès aux Donnés

Définition des règles d'accès

On présentera uniquement ce que l'on va mettre en œuvre (ex: "seul l'admin peut créer des fonctions" ne sera pas présenté.)

- Un étudiant ne peut que voir ses notes (SELECT)
- Les enseignants et personnels supérieurs tel que le directeur ou autre doivent pouvoir voir toutes les informations concernant les élèves
- Un enseignant peut agir sur les notes si il enseigne le groupe en question sur ce module (INSERT, DELETE, UPDATE)
- L'enseignant peut agir sur les contrôles seulement si il est responsable du module (INSERT, DELETE, UPDATE)
- Seul le directeur doit pouvoir agir sur les rôles par exemple insérer un nouvel enseignant/étudiant (INSERT, DELETE, UPDATE)

Procédures pour mettre en oeuvre ces règles

Afin que les étudiants, enseignants et supérieurs puissent utiliser les fonctions qui agissent sur les tables nous avons tout d'abord créé des ROLE etudiant, enseignant et superieur. À chaque ajout d'un étudiant dans la table etudiant un utilisateur est créé en lui attribuant le ROLE etudiant. De même pour les enseignants et supérieurs. Ceci est possible après que l'admin ai ajouté la personne dans la table personne.

```
CREATE ROLE enseignant;

CREATE ROLE superieur;

CREATE ROLE etudiant;

-- Ajout des droits sur les ROLE

GRANT SELECT ON TABLE evaluation, aFait, Personne, module, etudiant, competences,
possede TO etudiant;
```

```
GRANT SELECT ON TABLE evaluation, aFait, Personne, module, etudiant, enseignant,
competences, possede, superieur, dispense TO enseignant;

GRANT SELECT ON TABLE evaluation, aFait, Personne, module, etudiant, competences,
possede, superieur, dispense TO superieur;

GRANT INSERT, DELETE, UPDATE ON TABLE aFait, evaluation TO enseignant;

GRANT INSERT, DELETE, UPDATE ON TABLE Etudiant, Enseignant, Superieur TO superieur;

-- Trigger qui ajoute la personne au bon ROLE

CREATE OR REPLACE FUNCTION trigger_role() RETURNS TRIGGER AS

$$

   DECLARE

       user_name varchar;

   BEGIN

       IF TG_relname = 'etudiant' THEN

           EXECUTE 'CREATE USER "' || NEW.nip || '";';

           EXECUTE 'GRANT etudiant TO "' || NEW.nip || '";';

       ELSIF TG_relname = 'enseignant' THEN

           SELECT nom_personne INTO user_name FROM PERSONNE WHERE id_personne =
NEW.id_enseignant;

           EXECUTE 'CREATE USER "' || user_name || '";';

           EXECUTE 'GRANT enseignant TO "' || user_name || '";';

       ELSIF TG_relname = 'superieur' THEN

           SELECT nom_personne INTO user_name FROM PERSONNE WHERE id_personne =
NEW.id_superieur;

           EXECUTE 'CREATE USER "' || user_name || '";';

           EXECUTE 'GRANT superieur TO "' || user_name || '";';

       END IF;
```

```
      RETURN NEW;
   END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER give_etud
   AFTER INSERT ON etudiant
   FOR EACH ROW EXECUTE PROCEDURE trigger_role();
CREATE TRIGGER give_enseign
   AFTER INSERT ON enseignant
   FOR EACH ROW EXECUTE PROCEDURE trigger_role();
CREATE TRIGGER give_sup
   AFTER INSERT ON superieur
   FOR EACH ROW EXECUTE PROCEDURE trigger_role();
```

La suite présente la création des fonctions qui définissent les règles d'accès.

```
---------------------------------Accès Données Étudiant-------------------------------
-- L'étudiant peut voir que ses notes à lui
CREATE OR REPLACE FUNCTION mes_notes() RETURNS TABLE(module_ varchar, controle
varchar, meilleureNote decimal, note decimal, pireNote decimal) AS
$$
   BEGIN
      IF session_user IN (SELECT nip::name FROM Etudiant) THEN
          RETURN QUERY
              SELECT module, nom_controle, meilleure_note, note_, pire_note FROM
releve_de_note(session_user::integer);
      ELSE
```

```
            RAISE EXCEPTION 'Vous n''êtes pas étudiant de cette promo';
        END IF;
    END
$$ LANGUAGE plpgsql;
-- L'étudiant peut voir que son bulletin à lui
CREATE OR REPLACE FUNCTION mon_bulletin() RETURNS TABLE(competence varchar, moyenne
decimal) AS
$$
    DECLARE
        grp varchar;
    BEGIN
        IF session_user IN (SELECT nip::name FROM Etudiant) THEN
            SELECT groupe INTO grp FROM Etudiant WHERE nip = session_user::integer;
            RETURN QUERY
                SELECT comp, moy FROM bulletin(grp) WHERE nip_ = session_user::integer;
        ELSE
            RAISE EXCEPTION 'Vous n''êtes pas étudiant de cette promo';
        END IF;
    END
$$ LANGUAGE plpgsql;
-- L'étudiant peut voir que ses moyennes à lui
CREATE OR REPLACE FUNCTION mes_moyennes() RETURNS TABLE(module_ varchar,
meilleure_moyenne decimal, moyenne decimal, pire_moyenne decimal) AS
$$
    DECLARE
```

```
      grp varchar;
   BEGIN
      IF session_user IN (SELECT nip::name FROM Etudiant) THEN
          SELECT groupe INTO grp FROM Etudiant WHERE nip = session_user::integer;
          RETURN QUERY
              SELECT module, max_moyenne, moyenne_etudiant, min_moyenne FROM
moyenne_etudiant(grp) WHERE nip_ = session_user::integer ;
      ELSE
          RAISE EXCEPTION 'Vous n''êtes pas étudiant de cette promo';
      END IF;
   END
$$ LANGUAGE plpgsql;
-- L'étudiant peut voir son classement avec sa moyenne générale
CREATE OR REPLACE FUNCTION mon_classement() RETURNS TABLE(moyenne_ decimal,
classement_ integer) AS
$$
   BEGIN
      IF session_user IN (SELECT nip::name FROM Etudiant) THEN
          RETURN QUERY
              SELECT moyenne, classement FROM classement() WHERE nip_ =
session_user::integer ;
      ELSE
          RAISE EXCEPTION 'Vous n''êtes pas étudiant de cette promo';
      END IF;
   END
```

```
$$ LANGUAGE plpgsql;

-------------------------Accès Données Enseignant/Supérieur-------------------------

-- L'enseignant/supérieur peut voir les notes de l'étudiant qu'il veut

CREATE OR REPLACE FUNCTION notes_de(nip int) RETURNS TABLE(nom_etud varchar,
prenom_etud varchar, module_ varchar, controle varchar, note decimal) AS

$$

   BEGIN

       IF session_user::varchar IN (SELECT nom_personne FROM Personne JOIN Enseignant
ON id_personne = id_enseignant UNION SELECT nom_personne FROM Personne JOIN Superieur
ON id_personne = id_superieur) THEN

           RETURN QUERY

               SELECT nom, prenom, module, nom_controle, note_ FROM
releve_de_note(nip);

       ELSE

           RAISE EXCEPTION 'Vous n''êtes pas enseignant ou supérieur';

       END IF;

   END

$$ LANGUAGE plpgsql;

-- L'enseignant/supérieur peut voir le bulletin de l'étudiant qu'il veut

CREATE OR REPLACE FUNCTION bulletin_de(_nip int) RETURNS TABLE(nom_etud varchar,
prenom_etud varchar, competence varchar, moyenne decimal) AS

$$

   DECLARE

       grp varchar;

   BEGIN
```

```
      IF session_user::varchar IN (SELECT nom_personne FROM Personne JOIN Enseignant
ON id_personne = id_enseignant UNION SELECT nom_personne FROM Personne JOIN Superieur
ON id_personne = id_superieur) THEN

          SELECT groupe INTO grp FROM Etudiant WHERE nip = _nip;

          RETURN QUERY

              SELECT nom, prenom, comp, moy FROM bulletin(grp) WHERE nip_ = _nip;

      ELSE

          RAISE EXCEPTION 'Vous n''êtes pas enseignant ou supérieur';

      END IF;

   END

$$

LANGUAGE plpgsql;

-- L'enseignant/supérieur peut voir les moyennes de l'étudiant qu'il veut

CREATE OR REPLACE FUNCTION moyenne_de(_nip int) RETURNS TABLE(nom_etud varchar,
prenom_etud varchar, module_ varchar, moyenne decimal) AS

$$

   DECLARE

      grp varchar;

   BEGIN

      IF session_user::varchar IN (SELECT nom_personne FROM Personne JOIN Enseignant
ON id_personne = id_enseignant UNION SELECT nom_personne FROM Personne JOIN Superieur
ON id_personne = id_superieur) THEN

          SELECT groupe INTO grp FROM Etudiant WHERE nip = _nip;

          RETURN QUERY

              SELECT nom, prenom, module, moyenne_etudiant FROM moyenne_etudiant(grp)
WHERE nip_ = _nip;
```

```
      ELSE
          RAISE EXCEPTION 'Vous n''êtes pas enseignant ou supérieur';
      END IF;
   END
$$
LANGUAGE plpgsql;
-- Trigger pour que seul l'enseignant résponsable du module puisse
ajouter/modifier/supprimer des contrôles
CREATE OR REPLACE FUNCTION modif_eval()
RETURNS TRIGGER AS
$$
   DECLARE
      resp personne.nom_personne%TYPE;
   BEGIN
      SELECT nom_personne INTO resp
      FROM PERSONNE
      JOIN MODULE ON id_personne = id_enseignant
      WHERE MODULE.id_module = NEW.id_module;
      IF SESSION_USER = resp THEN
          RAISE NOTICE 'Modification avec succès !';
          RETURN NEW;
      ELSE
          RAISE EXCEPTION 'Vous n''êtes pas résponsable du module';
          RETURN NULL;
      END IF;
```

```
   END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER modif_eval
   BEFORE
   INSERT OR UPDATE OR DELETE ON EVALUATION
       FOR EACH ROW
       EXECUTE PROCEDURE modif_eval();
-- Trigger pour que seul l'enseignant du module et du groupe puisse
ajouter/modifier/supprimer les notes d'un étudiant
CREATE OR REPLACE FUNCTION trigger_note() RETURNS TRIGGER AS $$
BEGIN
   IF EXISTS (SELECT id_module, nom_personne, prenom_personne, dispense.groupe, nip
       FROM dispense JOIN personne On id_personne = id_enseignant
           JOIN Etudiant ON dispense.groupe = etudiant.groupe
           JOIN Evaluation USING(id_module)
               WHERE session_user = nom_personne::name AND NEW.id_etudiant = nip
                   AND NEW.id_eval = evaluation.id_eval) THEN
       RAISE NOTICE 'notes modifiées avec succès';
   ELSE
       RAISE EXCEPTION 'Permission non accordée';
       RETURN NULL;
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER modif_note

BEFORE

INSERT OR UPDATE OR DELETE

ON aFait

FOR EACH ROW

EXECUTE PROCEDURE trigger_note();
```